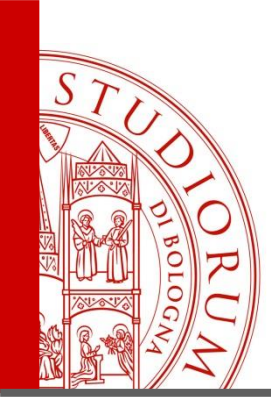


Università degli Studi di Bologna
Corso di Laurea in Ingegneria Informatica

Metadati e Introspezione

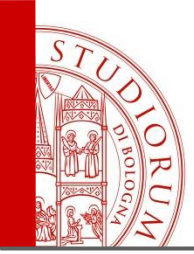
Ingegneria del Software T

Prof. MARCO PATELLA
Dipartimento di Informatica – Scienza e Ingegneria (DISI)

Metadati

"Metadata is data that describes other data. For example, the definition of a class is metadata"

Rumbaugh, J. et al, *Object Oriented Modeling and Design* [Prentice Hall, 1991]



Perché Metadati?

"Provided that a component comes with enough information to be self-describing, the interfaces supported by a component can be dynamically explored"

Szyperski, C.,
Component Software [Addison-Wesley, 1998]

Metadati C/C++

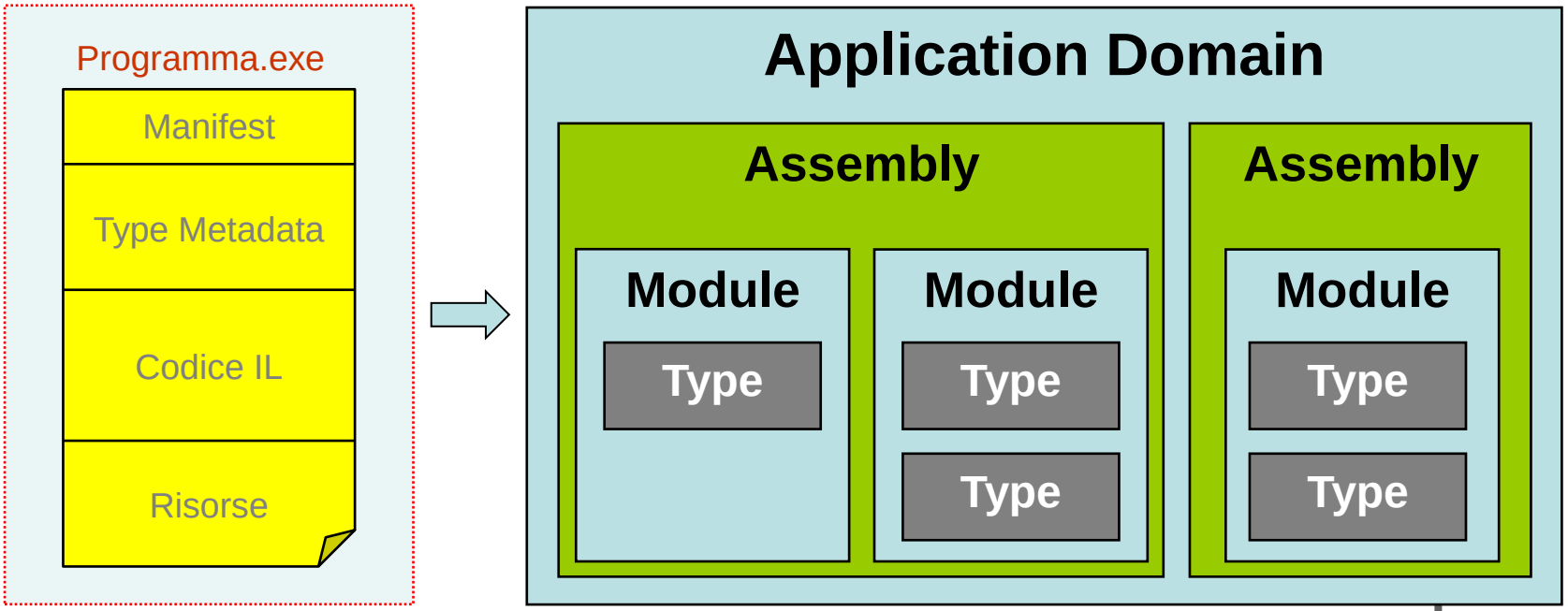
- Un **file header** C/C ++ può essere considerato **metadati**
- I client includono il file header in fase di compilazione per utilizzare i tipi che questo dichiara
- I file header C/C ++ sono **specifici del linguaggio**
- Fornire informazioni tra linguaggi diversi è un problema (molto) difficile

Metadati IDL

- COM e CORBA (Common Object Request Broker Architecture) utilizzano l'IDL (*Interface Definition Language*) per fornire i metadati
- Un **requisito aggiuntivo** da comprendere per gli sviluppatori
- Ospitati in **file separati** dal tipo che descrivono

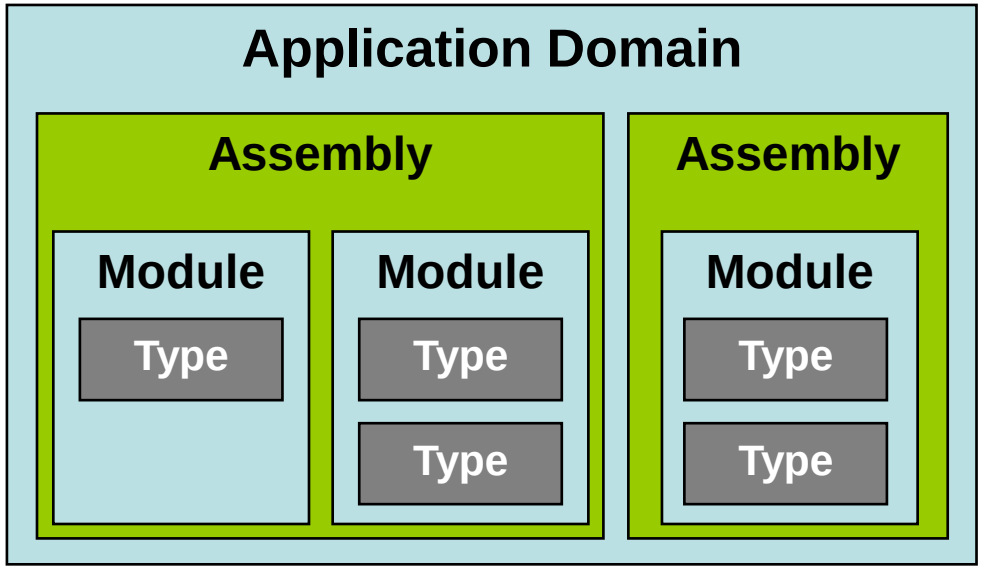
Metadati .NET (Java)

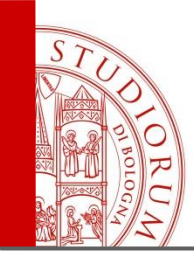
- Generati dalla definizione del tipo
- Memorizzati con la definizione del tipo
- Disponibili a tempo di esecuzione ► Reflection



Reflection

- La Reflection può essere utilizzata
  - **per esaminare i dettagli di un assembly**
  - **per istanziare oggetti e chiamare metodi conosciuti a tempo di esecuzione**
  - **per creare, compilare ed eseguire assembly «al volo»**





System.Type

- **`System.Type`** è il punto nodale per la reflection
  - Tutti gli oggetti e le variabili value sono istanze di tipi
  - È possibile scoprire il tipo di un oggetto o di un value

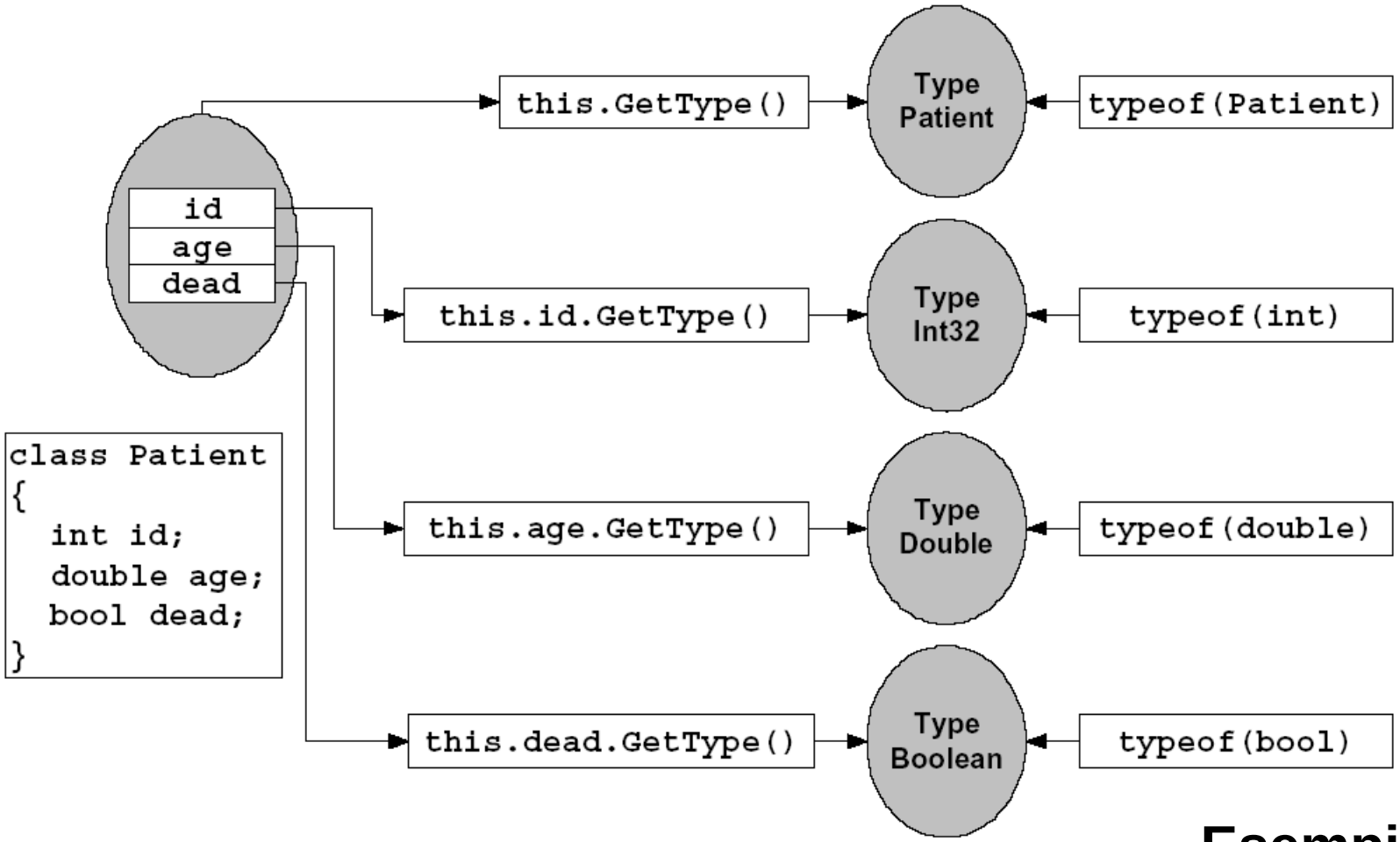
```
Type t0 = obj.GetType();
Type t1 = "Pippo".GetType();
```

  - È possibile referenziare il tipo per nome simbolico

```
Type t2 = typeof(System.String);
Type t3 = Type.GetType("System.String");
```

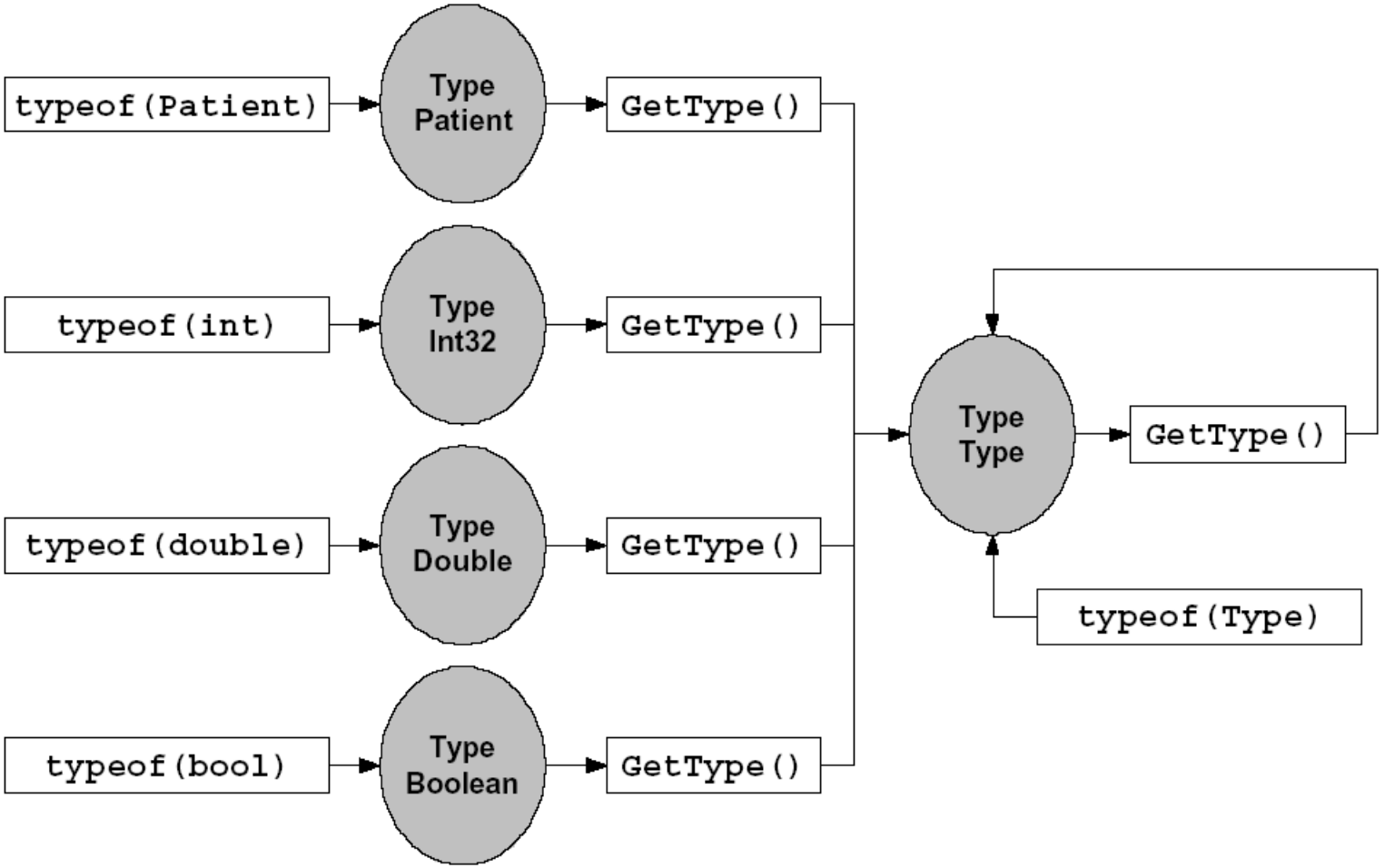
  - I tipi sono a loro volta istanze del tipo **`System.Type`**
- C'è un singolo oggetto **`Type`** per ciascun tipo definito nel sistema

System.Type

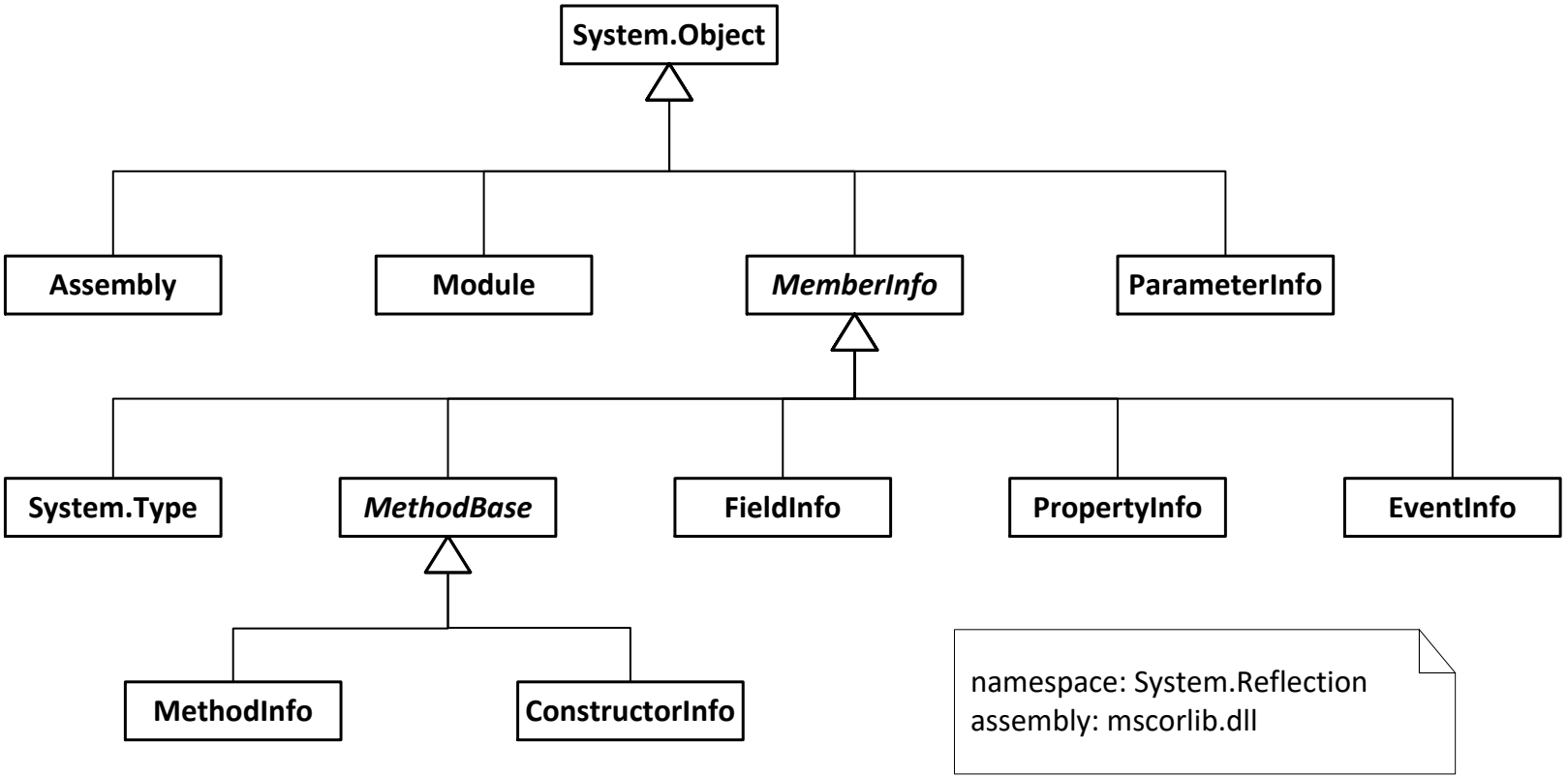


Esempio 5.1

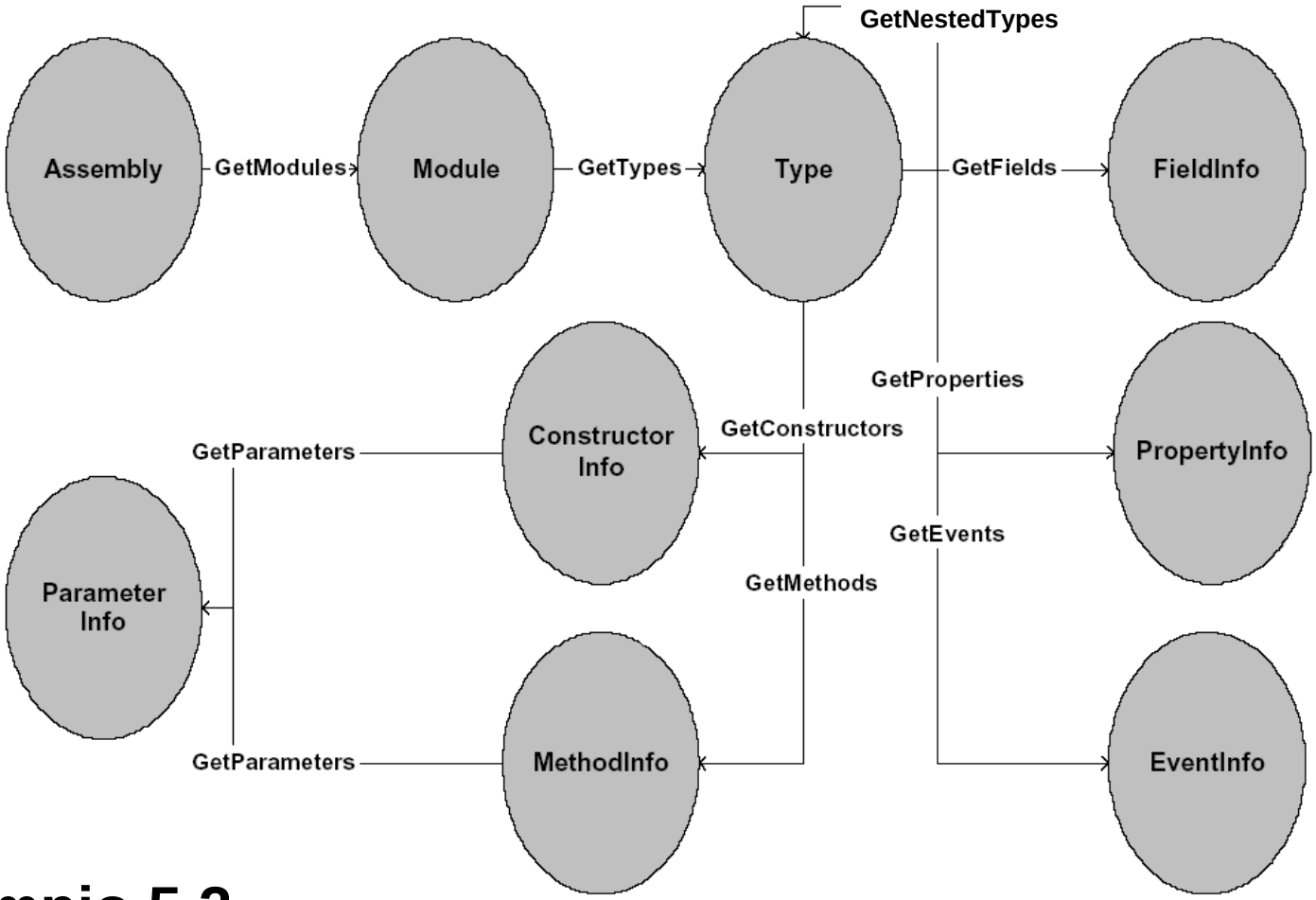
System.Type


typeof(Patient)
Type Patient
GetType()
typeof(int)
Type Int32
GetType()
typeof(double)
Type Double
GetType()
typeof(bool)
Type Boolean
GetType()
Type Type
GetType()
typeof(Type)

Il Modello a Oggetti della Reflection



Il Modello a Oggetti della Reflection

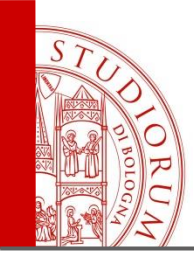


Esempio 5.2

Esempio
Enumerare tutti i Tipi Contenuti in un Assembly

1. **`Assembly.Load`** per caricare un assembly .NET restituisce un'istanza **`Assembly`**
2. **`Assembly.GetModules`** restituisce un array di **`Module`**
3. Per ogni **`Module`**, invocare **`Module.GetTypes`** restituisce un array di **`Type`**
4. Per ogni **`Type`**, …

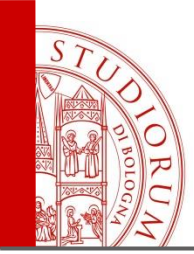
Esempio 5.3



Very late binding

- È possibile creare istanze dei tipi e/o accedere ai membri in modo **very late bound**
  - Si può istanziare il tipo in memoria, scegliendo il costruttore da chiamare
    - **`Activator.CreateInstance(type, …)`**
  - È possibile invocare metodi
    - **`methodInfo.Invoke(…)`**
  - È possibile invocare accessori e modificatori di proprietà
    - **`propertyInfo.GetValue(…)`**
    - **`propertyInfo.SetValue(…)`**
  - Membri pubblici sempre accessibili
  - Membri non pubblici accessibili se il chiamante dispone di autorizzazioni sufficienti

Esempio 5.4



System.Activator

- Crea istanze dinamicamente
- **`Activator.CreateInstance`** è l'equivalente late-bound dell'operatore **`new`**
  - Alloca lo spazio di archiviazione per la nuova istanza di tipo
  - Invoca il costruttore specificato
  - Restituisce un riferimento a un oggetto generico

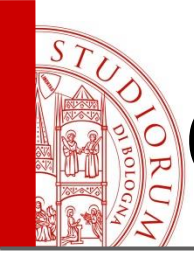
- **`T1 t = (T1) Activator.CreateInstance(typeof(T1));`**
- **`T1 t = (T1) Activator.CreateInstance(typeof(T1), object[] args);`**

Esempio 5.5

Attributi Personalizzati

- Sono un modo semplice per **aggiungere informazioni ai metadati** per qualsiasi elemento dell'applicazione
  - Possono essere applicati a un assembly usando una sintassi specifica
- Possono essere utilizzati in modo che **i clienti possano acquisire automaticamente determinate funzionalità**
  - Sono visibili tramite reflection
- Sono supportati in qualsiasi linguaggio .NET
- Sono in realtà solo **classi comuni** che derivano da **`System.Attribute`**
  - Possono contenere metodi e proprietà

AuthorAttribute



Creare Attributi Personalizzati

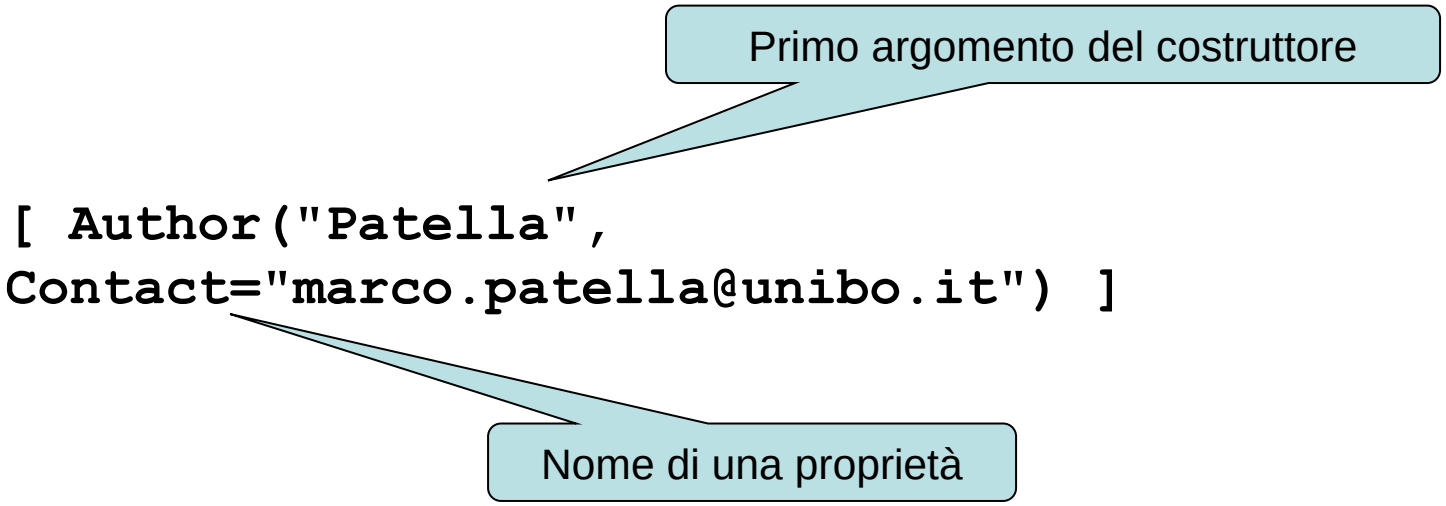
- Dichiarare la classe dell'attributo

```
public class AuthorAttribute : System.Attribute
```

- Dichiarare i costruttori
- Dichiarare le proprietà
- Applicare **`AttributeUsageAttribute`** (opzionale)
  Specifica alcune delle caratteristiche della classe
  - Il target dell'attributo (**`AttributeTargets`**)
  - A quali elementi l'attributo è applicabile
  - Indica se l'attributo può essere ereditato o meno (**`Inherited`**)
  - Se possono esistere o meno più istanze di un attributo per un elemento (**`AllowMultiple`**)

Usare Attributi Personalizzati

- C# usa sintassi IDL-like con [ ] prima della definizione del target
- Parametri per l'attributo passati
  - per **posizione** o
  - per **nome**



Accedere a Attributi Personalizzati

- Una volta creati gli attributi personalizzati, si usa la Reflection per leggerli
- È possibile ottenere un elenco di attributi personalizzati chiamando il metodo **`GetCustomAttributes`**

```
object[] X.GetCustomAttributes(inherit);
object[] X.GetCustomAttributes(attributeType,inherit);
```

`inherit` specifica se cercare nella catena di ereditarietà di questo membro per trovare gli attributi

- **`X`** è
  - Un'istanza di
    - **`Assembly, Module`**
    - **`MemberInfo`**
    - **`ParameterInfo`**

Esempio 5.6 – MainForm

Meta-Programmazione

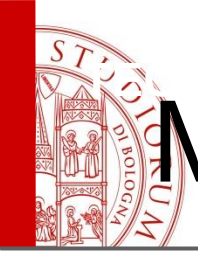
"... the fundamental problem is always the same:
preserve information available at compile time for inspection at runtime.
Making such information about a system available within that system
is called **reification**.
Programming a system to not only use reified information
but also to manipulate this information is called ***meta-programming***.
…*meta-programming* can be used to dynamically create new classes,
insert them into an existing inheritance graph and instantiate them"

Szyperski, C.,
Component Software [Addison-Wesley, 1998]

- **Reificazione**: Concretizzazione di un'astrazione

Meta-Programmazione in .NET

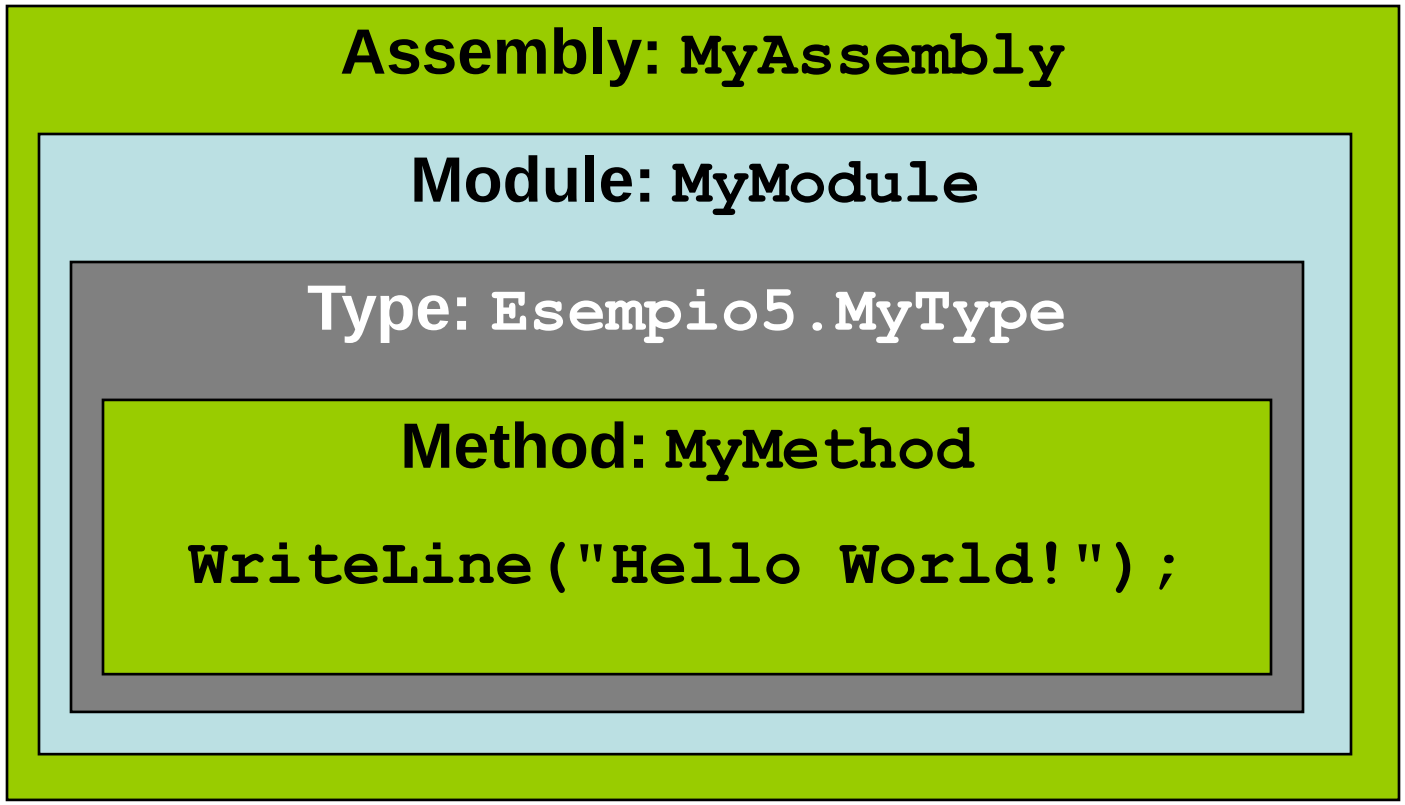
- Diverse classi funzionano insieme in .NET per raggiungere questo obiettivo
- Utilizzando gli oggetti precedenti e altri, è possibile **creare un assembly «al volo»**
  - **`Reflection.Emit`** consente di scrivere l'IL necessario per **creare e compilare l'assembly**
  - È quindi possibile **chiamare tale assembly** dal programma che lo ha creato
  - **L'assembly può essere memorizzato su disco** in modo che altri programmi possano utilizzarlo



Meta-Programmazione in .NET

- **`System.Reflection`**
  - **`AssemblyName`**
    Descrive completamente l'identità univoca di un assembly
- **`System.Reflection.Emit`**
  - **`AssemblyBuilder`**
    Definisce e rappresenta un assembly dinamico
  - **`ModuleBuilder`**
    Definisce e rappresenta un modulo
  - **`TypeBuilder`**
    Definisce e crea nuove istanze di classi a tempo di esecuzione
  - **`MethodBuilder`**
    Definisce e rappresenta un metodo (o costruttore) di una classe dinamica
  - **`ILGenerator`**
    Genera istruzioni del Microsoft intermediate language (MSIL)

Creazione Dinamica di un Tipo



Esempio 5.7